7

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[CS 230 Project Software Design Template 1](#_Toc923644072)

[Table of Contents 1](#_Toc2118870595)

[Document Revision History 2](#_Toc1236249716)

[Executive Summary 2](#_Toc1059281820)

[Requirements 3](#_Toc355911841)

[Design Constraints 3](#_Toc668535646)

[System Architecture View 3](#_Toc168972568)

[Domain Model 3](#_Toc2027019682)

[Evaluation 4](#_Toc214065429)

[Recommendations 4](#_Toc1479974146)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/27/2024 | Dat Nguyen | The initial creation of the software design document. This version includes a comprehensive outline and analysis of the project requirements for "Draw It or Lose It," covering server-side and client-side evaluations for multiple operating platforms (Mac, Linux, Windows, and Mobile Devices). Also included are detailed recommendations for the operating platform, system architecture, storage and memory management, distributed systems and networks, and security considerations tailored to the client's needs. Drafts for the Executive Summary, Requirements, Design Constraints, and System Architecture View have been formulated to guide the development and expansion of the game into a web-based multi-platform application. |
| 2.0 | 02/11/2024 | Dat Nguyen | This version introduces detailed evaluations and findings based on the client's request to expand "Draw It or Lose It" into a web-based multi-platform game. It includes an in-depth analysis of server-side hosting capabilities across Linux, Mac, Windows, and mobile platforms, highlighting Linux's scalability and cost-effectiveness as the recommended server operating system. The document further explores client-side development considerations, emphasizing the necessity of responsive design, cross-browser compatibility, and mobile device support to ensure a seamless user experience across all platforms. This update outlines the essential development tools, languages, and IDEs required for front-end and back-end development, assessing their impact on the development team and potential licensing costs. Recommendations for employing open-source tools to mitigate costs while ensuring efficient project execution have been added. The document also addresses the need for diversified skill sets within the development team, suggesting the division into specialized front-end, back-end, and QA teams to manage the project's scope effectively. |
| 3.0 | 02/25/2023 | Dat Nguyen | This version marks the improvement of the Recommendations section, providing a fully developed blueprint for expanding the game into a web-based multi-platform environment. This update integrates detailed recommendations for a Linux-based operating platform, highlighting its modular architecture, which is ideal for customizing the gaming application. It also elaborates on storage and memory management strategies tailored to the game's needs, utilizing distributed file systems and advanced memory management techniques to ensure scalability, high availability, and optimal performance. Further, the document outlines a comprehensive approach to distributed systems and network strategies, providing seamless communication and robust security measures across platforms. This version is designed to guide the development team in implementing a scalable, secure, and efficient infrastructure for "Draw It or Lose It," aligning with The Gaming Room's strategic goals for broadening their market reach and enhancing user experience. |

[Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is looking to develop a web-based game that serves multiple platforms based on their current game, "Draw It or Lose It," which is currently available only as an Android application. The primary goal is to extend the game's reach to multiple platforms, enhancing its accessibility and user engagement. The main challenge lies in transforming a single-platform game into a web-based application that is scalable, maintainable, and capable of supporting multiple teams and players. The current Android-only version needs the infrastructure and design to accommodate a multi-platform, web-based environment, necessitating a complete overhaul of the software architecture. The solution involves developing a robust web-based application using Java, adhering to software design patterns and principles. The design will focus on:

1. Singleton Pattern Implementation: To ensure a single instance of the game in memory, thus managing resources efficiently and preventing data conflicts.
2. Unique Identification: Utilizing unique identifiers for games, teams, and players to maintain data integrity and ease of access.
3. Iterator Pattern Usage: To facilitate unique naming for games and teams, enhancing user experience and database management.
4. Inheritance and Polymorphism: Implementing an Entity base class for common attributes, promoting code reusability, and simplifying future expansions.

The project will be executed in phases, starting with the software design document, followed by iterative development cycles. Continuous collaboration with The Gaming Room is essential to align the development with their vision and requirements. Regular reviews and adaptations based on feedback are crucial for the project's success. Our approach aims to deliver a scalable and efficient web-based version of "Draw It or Lose It" that aligns with the client's vision while adhering to industry-standard practices.

## Requirements

Business Requirements:

1. Multi-Platform Compatibility: The game must be accessible and functional across various platforms, not limited to the existing Android app.
2. User Engagement: The game should maintain or enhance the engagement levels of the current version, ensuring a seamless and enjoyable user experience.
3. Unique Team and Game Naming: Users can create uniquely named teams and games, enhancing personalization and user experience.

Technical Requirements:

1. Single Instance in Memory: Only one game instance should exist in memory at any given time, necessitating a design that effectively manages game instances.
2. Unique Identifiers: Implementation of unique identifiers for each game, team, and player to ensure data integrity and facilitate efficient data retrieval.
3. Inheritance and Polymorphism: A base Entity class should be created to hold common attributes and behaviors, with Game, Team, and Player classes inheriting from this base class.
4. Iterator Pattern for Name Checks: Employ the iterator pattern for adding games and teams to check for the uniqueness of names.
5. Robust Game Mechanics: The game should consist of four rounds of play, consistent image rendering, and a precise mechanism for team guessing and time management.
6. Software Design Patterns: Utilization of software design patterns like Singleton and Iterator to efficiently solve problems and manage resources.
7. Development Environment Setup: Assistance in setting up the development environment for the web-based version, as the client lacks this expertise.

## [Design Constraints](#_2et92p0)

1. Platform-Independent Architecture:
   1. Constraint: The application must be compatible across various platforms (browsers, operating systems).
   2. Implication: This necessitates a responsive design and careful selection of development frameworks and languages that support cross-platform compatibility.
2. Single Instance Management:
   1. Constraint: Only one game instance can exist in memory at any given time.
   2. Implication: Implementing the Singleton pattern is essential for managing game instances. This impacts memory management and data consistency, requiring careful instance creation and access handling.
3. Unique Identification System:
   1. Constraint: Unique identifiers must be assigned to each game, team, and player.
   2. Implication: This requires a robust and scalable ID generation and management system, impacting database design and query optimization.
4. Concurrency Control:
   1. Constraint: The application must handle multiple user interactions simultaneously without data conflicts.
   2. Implication: Implementing mechanisms for concurrency control, such as locking or transaction management, is crucial to prevent data corruption and ensure consistent user experience.
5. Responsive User Interface:
   1. Constraint: The UI must be intuitive and responsive, catering to different devices and screen sizes.
   2. Implication: This demands a flexible UI design, potentially employing frameworks like React or Angular, and rigorous testing across devices to ensure a consistent experience.
6. Network Latency and Scalability:
   1. Constraint: The web-based nature implies reliance on network connectivity, which can introduce latency.
   2. Implication: Efficient client-server communication and optimizing resource loading and data transfer are vital to minimize latency. Scalability must also be considered to handle varying loads.
7. Security Considerations:
   1. Constraint: Protecting user data and ensuring secure gameplay in a web environment.
   2. Implication: Implementing robust security measures, such as secure communication protocols (HTTPS), data encryption, and rigorous testing for vulnerabilities, becomes imperative.

## [System Architecture View](#_ilbxbyevv6b6)

**Architecture Components:**

1. Client Tier:
   1. Description: This tier comprises the user interface and client-side logic.
   2. Technologies: It could use HTML, CSS, and JavaScript, potentially with frameworks like React or Angular, for a responsive design.
   3. Function: Handles user interactions, displays game content, and communicates with the server tier.
2. Server Tier:
   1. Description: The backend logic and API endpoints reside here.
   2. Technologies: Java, using frameworks like Spring Boot for RESTful API development.
   3. Function: Processes game logic, manages game state, handles requests from the client tier, and communicates with the database tier.
3. Database Tier:
   1. Description: This tier handles data storage and retrieval.
   2. Technologies: SQL or NoSQL databases, depending on data structure and scalability needs.
   3. Function: Stores user, game, team, and player data, ensuring data integrity and providing efficient access.

Topology:

1. Communication Topology:
   1. Client-Server Communication: Utilizes HTTP/HTTPS protocols for secure data exchange.
   2. Real-time Data Handling: WebSockets or similar technology for real-time interaction and updates during gameplay.
   3. API Gateway: An API gateway could be employed for routing, monitoring, and securing API calls.
2. Storage Topology:
   1. Data Storage: Centralized database, ensuring data consistency and integrity.
   2. Backup and Recovery: Implementation of backup mechanisms and disaster recovery plans.
   3. Scalability: Considerations for scaling the database horizontally or vertically as per user load.
3. Security Architecture:
   1. Encryption: Secure data transmission using SSL/TLS encryption.
   2. Authentication and Authorization: Implementing OAuth, JWT, or similar protocols for secure access control.
   3. Data Privacy: Compliance with data privacy laws and regulations.

[Domain Model](#_8h2ehzxfam4o)"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.

The UML class diagram provided outlines the structure of the "The Gaming Room" application. It comprises several classes that interact to represent the domain of a web-based game.

1. Entity Class:
   1. This abstract class provides a foundation for all game objects with common attributes: id (a unique identifier) and name (a descriptive label for the Entity).
   2. Methods such as getId(), getName(), and toString() imply standard operations available to all entities, like retrieving the ID, name, and a string representation of the object.
2. Game, Team, and Player Classes:
   1. These concrete classes are inherited from the Entity class, meaning they all share the id, name attributes, and associated operations.
   2. The Game class includes a list of Team objects, signifying a one-to-many relationship where one game can have multiple teams.
   3. Similarly, the Team class holds a list of Player objects, representing the team members, again demonstrating a one-to-many relationship.
3. GameService Class:
   1. This class is designed as a singleton, indicated by the static service variable that holds its single instance and private constructor.
   2. It maintains lists of Game objects and provides methods to add and retrieve games and generate unique IDs, ensuring that each game, player, and team within the system is distinct.
4. ProgramDriver and SingletonTester Classes:
   1. ProgramDriver contains the main() method, serving as the application's entry point.
   2. SingletonTester is used to verify the singleton behavior of the GameService class.

Object-Oriented Principles:

1. Inheritance: Demonstrated by the Game, Team, and Player classes deriving from the Entity class. This allows for code reusability and a hierarchical structure where subclasses inherit base class attributes and methods.
2. Encapsulation: This is shown using the classes' private attributes and public methods. This principle hides the object's internal state and requires all interaction to be performed through public methods, protecting object integrity.
3. Polymorphism: This is implied through the toString() method, which can be overridden in subclasses of Entity to provide a class-specific string representation.
4. Abstraction: The Entity class provides a level of abstraction by defining a generic template for all entities within the game. This simplifies the system design by allowing the use of general types.
5. Singleton Pattern: The GameService class uses the singleton pattern to ensure a single point of management and access for the game services, optimizing resource use and ensuring consistency.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS is known for its stability and is a UNIX-based operating system, which makes it compatible with many server-side software used in web-based applications. Its advanced networking features provide a robust environment for hosting servers. The advantages include a user-friendly interface, powerful security features, and reliability. However, the cost of Mac hardware can be a significant disadvantage for large-scale deployments. MacOS is less widely used for servers than Linux or Windows, which may lead to a smaller community and less frequent updates for server-specific utilities. | Linux is famous for server-side applications due to its open-source nature, stability, flexibility, and security. It can be customized extensively to suit the specific needs of server operations and is supported by a vast community. Linux servers are known for their performance and are commonly used to host web applications, making them an excellent fit for web-based game hosting. The primary areas for improvement might be the steep learning curve for those unfamiliar with UNIX-like environments and potentially complex maintenance and troubleshooting processes. | Windows servers are known for their ease of use, integration with other Microsoft products, and support from Microsoft. They are often chosen for their compatibility with enterprise environments and applications that require .NET frameworks or other Windows-specific technologies. However, licensing costs can be high, and they may offer a different level of performance and customization than Linux servers. Windows servers also tend to be more vulnerable to malware than UNIX-based systems. | Hosting a web-based software application server-side on mobile devices is unconventional and not typically recommended due to their limited processing power, storage capacity, and network capabilities. Mobile devices are designed for client-side operations and consumption rather than serving web applications. They also need a more stable network connection and may face security issues, making them unsuitable for hosting server-side applications. |
| **Server-Based Deployment**: MacOS, known for its user-friendly desktop environment, is less commonly used as a server platform. However, it can serve web-based applications and offers a UNIX-based environment that supports various server software. It is more suitable for development or small-scale deployment scenarios.  **Licensing Costs**: MacOS comes with Apple hardware, and there are no additional costs for the operating system for server use. However, the initial investment in Apple hardware can be higher than comparable server hardware for Linux or Windows. This might impact the budget for server infrastructure. | **Server-Based Deployment**: Linux is highly regarded for its server capabilities and is widely used for web hosting and server infrastructure. It supports a wide range of server software, is known for its stability and security, and can quickly scale to provide thousands of players. Most web servers run on Linux due to its robustness and efficiency.  **Licensing Costs**: Linux has the advantage of being open source, which means there are no licensing costs for the operating system. This can significantly reduce the overall costs for The Gaming Room. However, costs may arise from using specific distributions that offer enterprise support, such as Red Hat Enterprise Linux (RHEL) or SUSE Linux Enterprise Server (SLES). | **Server-Based Deployment**: Windows Server is a widely used platform for hosting web-based applications, particularly those integrated with Microsoft technologies (.NET, Microsoft SQL Server, etc.). It offers robust support for enterprise-level applications and scalability to support large numbers of users.  **Licensing Costs**: The Windows Server operating system requires the purchase of a license, which can be a significant cost depending on the version and the required features (Standard, Datacenter, etc.). Additionally, client access licenses (CALs) may be needed for users or devices connecting to the server, increasing the cost. | **Server-Based Deployment**: Hosting server-side applications directly on mobile platforms is unconventional and not practical due to hardware, storage, and network limitations. Mobile devices are typically clients in web-based applications, not servers.  **Licensing Costs**: Not applicable for server-side deployment. Mobile platforms are not designed or licensed for hosting web-based application servers. |
| **Client Side** | When developing for Mac clients, considerations include the cost of Apple hardware and software licenses and the need for developers with expertise in Mac-specific technologies like Swift for native applications or web technologies for browser-based clients. Development time can be longer due to the closed nature of the Apple ecosystem, although the availability of development tools like Xcode provides a comprehensive environment for development. | Developing for Linux clients can be cost-effective due to the open-source nature of the operating system, leading to a wide range of free tools and libraries. However, ensuring compatibility across different Linux distributions can increase development time. Expertise in various programming languages and environments is necessary, as no standard Linux development platform exists. | Windows has a significant market share, making supporting it as a client platform essential. Development considerations include the cost of licenses for the Windows operating system and development tools, although free tools are available. Development time can be efficient due to the wide range of resources and support for Windows development, but expertise in Microsoft's development environments is required. | Development for mobile clients requires consideration of the diverse ecosystem, including multiple operating systems (iOS, Android) and device form factors. Costs can be high due to the need for various devices for testing, and development time may be extended due to the need to support multiple screen sizes and OS versions. Expertise is needed in mobile-specific programming languages, tools like Swift for iOS and Kotlin for Android, and an understanding of responsive web design for browser-based clients. |
| **Compatibility Requirements:**   * **Responsive Design**: A responsive web design is essential to ensure the application's interface automatically adjusts to different screen sizes and resolutions, providing an optimal viewing experience across various devices, from desktops to tablets and smartphones. * **Cross-Browser Compatibility**: The application must function seamlessly across various web browsers (e.g., Chrome, Firefox, Safari, Edge) on desktop and mobile platforms. This involves using standardized web technologies and performing thorough testing on different browsers to identify and fix compatibility issues. * **Mobile Device Support**: Given the diversity in mobile device capabilities, including varying screen sizes, resolutions, and operating system versions, the application must be designed with a mobile-first approach. This approach prioritizes the performance and usability on mobile platforms to ensure a consistent and engaging user experience.   **Development Considerations:**   1. **Cost**: 2. **Cross-Platform Development Tools**: Utilizing frameworks and tools that support cross-platform development (e.g., React, Angular, Vue.js for frontend; Node.js for backend) can reduce development costs by allowing code reuse across multiple platforms. 3. **Testing and Compatibility**: Investment in testing tools and services to ensure compatibility across browsers and devices can increase initial costs but is crucial for delivering a quality user experience. 4. **Time**: 5. **Development and Testing**: Developing a responsive, cross-browser, and cross-platform application requires additional design, implementation, and extensive testing to ensure compatibility and performance across all targeted devices and browsers. 6. **Continuous Updates**: The need to stay updated with the latest browser versions, operating system updates, and web standards can increase development time over the project's lifecycle. 7. **Expertise**: 8. **Skilled Developers**: A team with expertise in responsive web design, cross-platform development frameworks, and modern web technologies is necessary. Knowledge of accessibility and usability standards is also crucial to cater to a diverse user base. 9. **QA and Testing Expertise**: Quality assurance professionals experienced in cross-browser and cross-device testing are essential to identify and resolve compatibility issues during development. | | | |
| **Development Tools** | MacOS supports development with tools like Xcode for native applications and various web development IDEs for web-based applications, such as Atom, Sublime Text, and Visual Studio Code. Programming languages commonly used include Swift for native Mac applications and cross-platform web technologies like HTML, CSS, JavaScript, and server-side languages like Java and Python. | Linux supports various programming languages and tools for web-based application development. Familiar tools include Eclipse, NetBeans, and Visual Studio Code. Languages like Python, Java, JavaScript, PHP, and Ruby are widely supported, along with server technologies like Apache and Nginx. | Windows supports development with Microsoft Visual Studio, one of the most comprehensive IDEs, supporting languages like C#, VB.NET, and F#. Other IDEs like Visual Studio Code are also commonly used for web development. Windows is a robust platform for developing .NET applications and supports Java, Python, and other popular web development languages. | Development for mobile devices typically involves IDEs like Xcode for iOS development and Android Studio for Android development. The primary languages are Swift for iOS and Kotlin or Java for Android. Cross-platform development tools like Flutter and React Native are also used to build applications for both iOS and Android with a single codebase. For web-based applications, mobile browsers are targeted using responsive web design practices. |
| **Frontend Development:**   * **Languages**: HTML5, CSS3, and JavaScript are essential for creating a responsive and interactive web interface. * **Frameworks/Libraries**: React, Angular, or Vue.js can be used to develop dynamic and responsive user interfaces efficiently. These frameworks also support the mobile-first approach necessary for the application. * **IDEs/Editors**: Visual Studio Code, Sublime Text, or Atom are popular choices offering extensive web technologies and framework support.   **Backend Development**:   * **Languages**: JavaScript (Node.js), Python (Django or Flask), or Java (Spring Boot) are commonly used for server-side development. * **Database Management**: Depending on the application's data storage needs, SQL (PostgreSQL, MySQL) or NoSQL (MongoDB) databases. * **IDEs**: IntelliJ IDEA for Java, PyCharm for Python, and Visual Studio Code or WebStorm for Node.js.   **Mobile Development**:   * **Tools for Responsive Testing**: Chrome DevTools, Firefox Developer Edition, and tools like BrowserStack for cross-browser and device testing. * **Frameworks for Progressive Web Apps (PWA)**: Tools like Lighthouse to audit and improve the quality of PWAs for mobile platforms.   **Impact on the Development Team**:   * **Skillset Diversification**: Using various programming languages and frameworks requires a development team with diverse skills and expertise. Specialists in front-end, back-end, and mobile-responsive design are necessary. * **Collaboration and Integration**: Teams must collaborate effectively to integrate front-end and back-end systems, necessitating tools for version control (e.g., Git) and continuous integration/continuous deployment (CI/CD) pipelines. * **Multiple Teams Consideration**: Depending on the project's scale, separate teams for frontend, backend, and quality assurance might be needed to manage the workload and specialized tasks efficiently.   **Licensing Costs**:   * **IDEs and Tools**: Many IDEs like Visual Studio Code, Atom, and Sublime Text offer free versions that are highly capable of web development. However, some IDEs like IntelliJ IDEA and PyCharm have professional versions with additional features that require paid licenses. * **Frameworks/Libraries**: Most famous web development frameworks and libraries (e.g., React, Angular, Vue.js, Node.js) are open source and free to use. * **Testing and Deployment Tools**: Open-source tools are available for version control and CI/CD. However, cloud-based testing services like BrowserStack and deployment platforms (AWS, Google Cloud, Azure) may incur costs based on usage. | | | |

## Recommendations

I recommend adopting a Linux-based operating platform to expand "Draw It or Lose It" into various computing environments. Linux is renowned for its robustness, flexibility, and security, making it an ideal choice for developing and deploying a multi-platform web-based gaming application. As an open-source solution, Linux offers cost-effectiveness and scalability, which are crucial for server-side operations. Its compatibility with cloud environments further enhances its suitability for distributed systems that demand high scalability and reliability.

1. **Operating Systems Architectures:** Linux's architecture is inherently modular, centered around a monolithic kernel that manages core functions such as process control, memory management, and hardware interaction. Many userland tools and services surround this modular structure, enabling customized management of network connections, application processes, and user interfaces. This adaptability permits an environment finely tuned for "Draw It or Lose It," accommodating various deployment scenarios like cloud-based infrastructures, dedicated servers, or hybrid configurations, thus optimizing performance and resource utilization.
2. **Storage Management:** Considering the Linux platform's versatility, a storage management solution integrating distributed file systems and object storage is advisable. Technologies like GlusterFS or Ceph are recommended for their scalability and fault tolerance, vital attributes for a game necessitating extensive data and media asset storage. Utilizing robust systems such as PostgreSQL or MongoDB for database storage will facilitate effective data management and support complex gaming system requirements through advanced query capabilities.
3. **Memory Management:** Linux exhibits superior memory management through an efficient multi-level page cache system, enhancing access speed to frequently used data while minimizing the footprint of less accessed data. Features like Transparent HugePages and Kernel SamePage Merging (KSM) further refine memory utilization and decrease latency, which is critical for ensuring the smooth operation of a real-time application like "Draw It or Lose It."
4. **Distributed Systems and Networks:** To facilitate seamless communication across different platforms for "Draw It or Lose It," a distributed architecture leveraging RESTful APIs for cross-platform interoperability and WebSockets for real-time communication is recommended. Utilizing load balancers and Content Delivery Networks (CDNs) will aid in managing network traffic and minimizing latency, maintaining consistent application performance even during peak user activity. A comprehensive monitoring and fault tolerance plan is essential to address potential connectivity issues or outages within the distributed network, ensuring uninterrupted game availability.
5. **Security:** Linux's security model, encompassing discretionary and mandatory access controls (e.g., SELinux) and advanced firewalling techniques (iptables, nftables), provides a solid foundation for securing "Draw It or Lose It." Implementing HTTPS with Transport Layer Security (TLS) is crucial for safeguarding data during transmission. Data storage security should include encryption for data at rest and in transit, protecting sensitive user information. Regular security audits, utilizing intrusion detection systems, and adhering to a strict security patching regime are imperative measures. Adherence to security standards and best practices, such as those outlined by OWASP for web applications, will ensure comprehensive protection against prevalent security threats.

The recommended Linux-based platform, supported by a detailed analysis of operating system architectures, storage, and memory management strategies, alongside a robust approach to distributed systems, networks, and security, provides a comprehensive blueprint for expanding "Draw It or Lose It" to various computing environments. This recommendation not only aligns with The Gaming Room's technical requirements and operational goals but also positions "Draw It or Lose It" for sustainable growth and success in the competitive gaming market.